Projeto ABM para Simulação de Dengue

Guia para Implementação e Estudo

Baseado em Mesa (Python)

31 de julho de 2025

Resumo

Este guia tem como objetivo apresentar os passos necessários para utilizar a biblioteca *Mesa* na simulação da propagação da dengue em uma população composta por seres humanos e mosquitos. A proposta consiste em coletar dados epidemiológicos dos últimos anos (por exemplo, de 2015 a 2024), construir um modelo baseado em *Agentes* utilizando as informações entre o primeiro e o penúltimo ano, e então avaliar a capacidade do modelo de reproduzir o que ocorreu no último ano, servindo como um teste de validação retrospectiva.

Além disso, a simulação poderá ser executada múltiplas vezes, introduzindo variações aleatórias nos parâmetros dentro de faixas plausíveis. A partir dessas execuções, é possível calcular um *intervalo de confiança* (IC) que represente a incerteza inerente ao modelo e, com base nesses resultados, ajustar e definir os parâmetros mais adequados para a simulação.

1 Estrutura de Pastas do Projeto

A estrutura abaixo organiza bem o projeto e separa os dados, o código e as análises:

```
dengue_abm_project/
│
├── data/
│   ├── raw/          # Dados brutos (ex: notificações, chuva, temperatura)
│   └── processed/    # Dados limpos e formatados para o modelo
│
├── model/
│   ├── agents.py     # Classes dos agentes (Humanos e Mosquitos)
│   ├── model.py      # Classe principal do modelo ABM de dengue
│   └── parameters.py # Configurações e parâmetros do modelo
│
├── analysis/
│   └── evaluate_model.py  # Avaliar a acurácia do modelo vs dados reais
│
├── visualization/
│   └── plot_results.py    # Scripts para gerar gráficos e mapas
│
├── notebooks/
│   ├── 01_explore_data.ipynb  # Exploração dos dados (EDA)
│   └── 02_run_simulation.ipynb# Simulação e análise de resultados
│
├── results/          # Resultados de simulações (ex: CSVs, gráficos)
├── docs/             # Documentação extra (PDFs, notas)
└── README.md         # Resumo do projeto
```

2 Código Base dos Agentes (agents.py)

```python
from mesa import Agent
import random

class HumanAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
```

```
7           self.state = "S"  # Estados: S (susceptível), I (
                infectado), R (recuperado)
8           self.days_infected = 0
9
10      def step(self):
11          if self.state == "I":
12              self.days_infected += 1
13              if self.days_infected > self.model.recovery_time:
14                  self.state = "R"
15
16  class MosquitoAgent(Agent):
17      def __init__(self, unique_id, model):
18          super().__init__(unique_id, model)
19          self.state = "S"
20
21      def step(self):
22          if self.state == "I":
23              humans = [a for a in self.model.grid.
                    get_cell_list_contents([self.pos])
24                        if isinstance(a, HumanAgent) and a.
                              state == "S"]
25              for human in humans:
26                  if random.random() < self.model.
                        transmission_prob:
27                      human.state = "I"
28          else:
29              humans = [a for a in self.model.grid.
                    get_cell_list_contents([self.pos])
30                        if isinstance(a, HumanAgent) and a.
                              state == "I"]
31              if humans and random.random() < self.model.
                    transmission_prob:
32                  self.state = "I"
```

3 Código Base do Modelo (`model.py`)

```
from mesa import Model
from mesa.time import RandomActivation
from mesa.space import MultiGrid
from model.agents import HumanAgent, MosquitoAgent

class DengueABM(Model):
    def __init__(self, width=10, height=10, initial_humans=50, initial_mosquitoes=100,
                 transmission_prob=0.2, recovery_time=14):
        super().__init__()
        self.grid = MultiGrid(width, height, torus=True)
        self.schedule = RandomActivation(self)
        self.transmission_prob = transmission_prob
        self.recovery_time = recovery_time

        for i in range(initial_humans):
            human = HumanAgent(i, self)
            self.schedule.add(human)
            x, y = self.random.randrange(width), self.random.randrange(height)
            self.grid.place_agent(human, (x, y))

        for i in range(initial_humans, initial_humans + initial_mosquitoes):
            mosquito = MosquitoAgent(i, self)
            self.schedule.add(mosquito)
            x, y = self.random.randrange(width), self.random.randrange(height)
            self.grid.place_agent(mosquito, (x, y))

    def step(self):
        self.schedule.step()
```

4 Intervalo de Confiança de 95% (IC 95%)

O IC 95% indica a faixa onde esperamos que o valor verdadeiro esteja com 95% de confiança:

$$IC = \bar{x} \pm 1.96 \cdot \frac{\sigma}{\sqrt{n}}$$

4.1 Exemplo em Python

```python
import numpy as np
import matplotlib.pyplot as plt

resultados = np.random.normal(loc=5000, scale=400, size=100)
media = np.mean(resultados)
desvio = np.std(resultados, ddof=1)
n = len(resultados)
ic_inf = media - 1.96 * (desvio / np.sqrt(n))
ic_sup = media + 1.96 * (desvio / np.sqrt(n))

print(f"Média: {media:.1f}")
print(f"IC 95%: [{ic_inf:.1f}, {ic_sup:.1f}]")

plt.hist(resultados, bins=15, alpha=0.7)
plt.axvline(ic_inf, color='red', linestyle='--')
plt.axvline(ic_sup, color='red', linestyle='--')
plt.axvline(media, color='green')
plt.show()
```

Este gráfico mostra a distribuição das simulações com os limites do IC 95%.